Lehrstuhl für Informatik 3
Rheinisch-Westfälische Technische Hochschule
Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Aachen, 13. 04. 2007

Nagl, Ranger, Wörzberger

Übungen zur Vorlesung
„Grundgebiete der Informatik 2:
Algorithmen und Programmiertechniken“

— Lösung zu Blatt 1 —

1. Aufgabe

(a) Einfache Beispiele für die EBNF-Konstrukte außer Rekursion finden sich unten. Rekursion kommt bei Ausdrücken und Statements vor (*expression*, *statement*), aber nur über mehrere Ebenen. Aus Platzgründen werden sie daher hier nicht wiedergegeben. Der Grund dafür ist, daß die Syntaxbeschreibung Produktionen inhaltlich gruppiert und —insbesondere bei Ausdrücken— die Operator-Präzedenzen beinhaltet.

Sequenz:	*operator_symbol*::=operator *operator*
	typeid_expression::=typeid (*expression*)
	inc_statement::=*designator* ++
Option:	*throw_statement*::=throw **[** *expression* **]**
	enumerator_definition::=*identifier* **[** = *constant_expression* **]**
	member_initializer::=*identifier* (**[** *expression_list* **]**)
Wiederholung:	*declaration_list*::=**{** *declaration* ; **}**
	inheritance_spec::=: *inheritance* **{** , *inheritance* **}**
	program_file::=**{** *context_clause* **}** **{** *declaration* **}**$^+$
Alternative:	*boolean_literal*::=true **|** false
	inc_dec::=*inc_statement* **|** *dec_statement*
	file::=*program_file* **|** *header_file*

ifStatement $\xrightarrow{(1)}$ if (*expression*) { *statementList* }
$\xrightarrow{(2),(6)}$ if (*simpleExpression*) { *statement* }
$\xrightarrow{(4),(8)}$ if (*factor*) { cout << *string* ; }
$\xrightarrow{(5)}$ if (*ident*) { cout << *string* ; }

ifStatement $\xrightarrow{(1)}$ if (*expression*) { *statementList* }
else if (*expression*) { *statementList* }
else { *statementList* }

$\xrightarrow{(2)^2,(6)^3}$ if (*simpleExpression relation simpleExpression*)
{ *statement* }
else if (
simpleExpression relation simpleExpression)
{ *statement* }
else { *statement* }

$\xrightarrow{(3)^2,(4)^4,(8)^3}$ if (*factor addOperator factor* == *factor*)
{ cout << *string* ; }
else if (*factor addOperator factor* < *factor*)
{ cout << *string* ; }
else { cout << *string* ; }

$\xrightarrow{(5)^6,(7)^2}$ if (*ident* + *ident* == *number*)
{ cout << *string* ; }
else if (*ident* + *ident* < *number*)
{ cout << *string* ; }
else { cout << *string* ; }

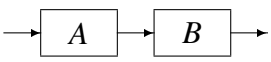
(b) Die folgende Grammatik enthält die nötigen Änderungen in den Regeln (4) und (5).

ifStatement	::=	if (*expression*) { *statementList* } **{** else if (*expression*) { *statementList* } **}** **[** else { *statementList* } **]**	(1)
expression	::=	*simpleExpression* **[** *relation simpleExpression* **]**	(2)
relation	::=	== | != | < | <= | > | >=	(3)
simpleExpression	::=	**[** + | – **]** *term* **{** *addOperator term* **}**	(4)
term	::=	*factor* **{** (* | /) *factor* **}**	(5)
factor	::=	*number* | *ident* | (*expression*) | ! *factor*	(6)
statementList	::=	*statement* **{** *statement* **}**	(7)
addOperator	::=	+ | – | || | &&	(8)
statement	::=	*ifStatement* | cout << *string* ;	(9)

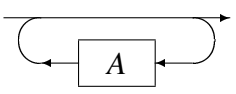
2. Aufgabe

(a) Syntaxdiagramme für EBNF-Konstrukte:

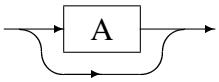
Sequenz(*A B*):



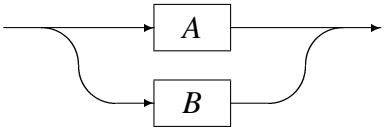
Wiederholung({ *A* }):



Option([*A*]):



Alternative(*A* | *B*):



Für die Rekursion kann kein passendes Syntaxdiagramm angegeben werden, sie entsteht durch Verwendung eines Nichtterminalsymbols, das in einem Syntaxdiagramm definiert wird, auf der rechten Seite derselben Definition (direkte Rekursion). Dies kann auch durch eine zyklische Kette von Definitionen geschehen (indirekte Rekursion), also z.B. das Nichtterminalsymbol A wird mit Hilfe von B definiert und B wird mit Hilfe von A definiert.

(b) Übersetzung der Syntaxdiagramme in eine Grammatik:

$$
\begin{aligned}
\textit{teilchen} &::= \quad \{\ \textit{ladung}\ \}^{+} && (1)\\
\textit{ladung} &::= \quad (\ +\textit{negativ}\ ) \mid (\ -\textit{positiv}\ ) && (2)\\
\textit{positiv} &::= \quad + \mid (\ -\textit{positiv}\ \textit{positiv}\ ) && (3)\\
\textit{negativ} &::= \quad - \mid (\ +\textit{negativ}\ \textit{negativ}\ ) && (4)
\end{aligned}
$$

(c) Die gleiche Sprache, nur mit weniger Regeln:

$$
\begin{aligned}
\textit{teilchen} &::= \quad \{\ (\ +\textit{negativ}\ ) \mid (\ -\textit{positiv}\ )\ \}^{+} && (1)\\
\textit{positiv} &::= \quad + \mid (\ -\textit{positiv}\ \textit{positiv}\ ) && (2)\\
\textit{negativ} &::= \quad - \mid (\ +\textit{negativ}\ \textit{negativ}\ ) && (3)
\end{aligned}
$$

3. Aufgabe

(a) Das klassische Programm:

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int main()
{
   cout << "Mein Name" << endl;
   return 0;
}
```

(b) Siehe (c).

(c) Natürlich etwas überstrukturiert:

Implementation `main.cc`

```
#include "stdafx.h"
#include "summe.h"
#include <iostream>
using namespace std;

int main()
{
   int geleseneZahl;            // Speicher fuer die Zahl
   cin >> geleseneZahl;         // Erste Zahl einlesen
   while (!cin.eof()) {         // Schleife bis Eingabe leer
      summeDazu(geleseneZahl);  // aufaddieren lassen
      cin >> geleseneZahl;      // Weitere Zahl einlesen
   }
```

```
   // Ausgeben lassen.
   cout << "Gesamtsumme: " << summeWert() << endl;
   return 0;
}
```

Schnittstelle `summe.h`

```
#pragma once

#include "stdafx.h"

/* Addiere den neuen Wert zur aktuellen Summe hinzu.  */
void summeDazu(int neuerWert);
/* Liefert die aktuelle Summe.  Anfangs ist der Wert 0. */
int summeWert(void);
```

Implementation `summe.cc`

```
#pragma once

#include "stdafx.h"
#include "summe.h"
/* Der Datenspeicher fuer die aktuelle Summe. */
static int aktuelleSumme = 0;

void summeDazu(int neuerWert)
{
   aktuelleSumme += neuerWert;
}

int summeWert(void)
{
   return aktuelleSumme;
}
```

(d) Eine komprimierte Lösung ist:

```
#include <iostream>
#include <ctime>
using namespace std;
int main() // Ein Uebungsprogramm, das die Uhrzeit ausgibt.
{
   time_t sekundenSeit;         // Sekunden seit 1970
   sekundenSeit = time(NULL);   // Anzahl ermitteln
   cout << "Sekunden seit Anfang 1970: " << sekundenSeit << endl;
   /* localtime wertet die Zeitzone aus,
      asctime stellt das als Zeichenkette dar. */
   cout << "Anders gesagt: " << asctime(localtime(&sekundenSeit))
        << endl;
   return 0;
}
```